Synet : un outil de synthèse de réseaux de Petri bornés, applications

Benoit Caillaud

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

SYNET : *un outil de synthèse de réseaux de Petri bornés, applications*

Benoît Caillaud

N° 3155

avril 1997

THÈME 1

ISSN 0249-6399

SYNET : un outil de synthèse de réseaux de Petri bornés, applications

Benoît Caillaud

Thème 1 — Réseaux et systèmes
Projet Pampa

Rapport de recherche n° 3155 — avril 1997 — 25 pages

Résumé : SYNET est un outil de synthèse de réseaux de Petri bornés à partir de systèmes de transition finis. Il met en œuvre des algorithmes polynomiaux de calcul de régions dans les systèmes de transition et repose sur des méthodes de programmation linéaire dans les nombres rationnels. Un nouvel algorithme de synthèse de réseaux répartissables dans lequel la synthèse est contrainte par un placement des actions sur un ensemble de procesus répartis y est également mis en œuvre. Les réseaux alors produits peuvent être exécutés dans un environnement réparti. L'application de SYNET à la synthèse d'un protocole de communication (connexion-déconnexion) à partir d'une spécification de service est ensuite détaillée.

Mots-clé : réseaux de Petri, régions dans les systèmes de transition, synthèse de réseaux, programmation linéaire, protocoles de communication, répartition automatique, méthodologie de programmation répartie.

(Abstract: pto)

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : (33) 02 99 84 71 00 – Télécopie : (33) 02 99 84 71 71

SYNET: A Tool for the Synthesis of Bounded Petri-Nets, Applications

Abstract: SYNET is a tool synthesizing bounded Petri-nets from finite transition systems. The algorithms used are polynomial and consist in a computation of regions in transition systems based on linear programming methods on rationnal numbers. SYNET also implements a new algorithm for the synthesis of distribuable nets in which events are mapped on a set of distributed processes. Then, the synthesized nets can be executed on a distributed environment. Finally, an application of SYNET is reviewed: the synthesis of a connection-disconnection communication protocol from its specification of service.

Key-words: Petri-nets, Regions in Transition Systems, Net-Synthesis, Linear Programming, Communication Protocols, Automatic Distribution, Distributed Programming Methodology

Table des matières

1 Introduction

Les régions dans les langages et les systèmes de transition est le concept fondateur des méthodes de synthèse de réseaux [9, 2]. Ce concept est à l'origine d'algorithmes polynomiaux pour la synthèse de réseaux de Petri bornés [1] et de réseaux flip-flop, variante des réseaux élémentaires [20].

Pour un système de transition fini donné, le problème de la synthèse d'un réseau est de décider constructivement de l'existence d'un réseau dont le graphe des marquages est isomorphe au système de transition de départ.

Le problème de synthèse peut également être défini pour des langages: il s'agit alors de décider de l'existence d'un réseau dont le langage est donné sous la forme d'une expression régulière, et dans l'affirmative de le calculer.

Même si on ne peut contester l'intérêt de ces résultats, les applications de ces techniques sont aujourd'hui extrêmement limitées. Dans [7, 6], un calcul des régions est utilisé pour la génération de circuits VLSI asynchrones.

Un autre domaine d'applications potentielles est la répartition d'automates réactifs, utile à l'exécution répartie de programmes réactifs synchrones et à la synthèse de protocoles de communication — voir [14, 5].

Les techniques de synthèse de réseaux permettent une extraction de la concurrence présente dans les systèmes de transition finis — présence de losanges. Elles constituent de ce fait un outil potentiel pour la répartition de programmes séquentiels. Il faut toutefois être conscient de leurs limitations, qui sont de deux ordres :

- tout système de transition n'est pas le graphe des marquages d'un réseau dans lequel chaque action n'est l'étiquette que d'une seule transition. Ce fait est alors caractérisé par l'échec d'un problème de séparation.

- À cause des conflits, tout réseau n'est pas aisément exécutable en réparti.

C'est pour permettre l'étude et l'adaptation de ces techniques aussi bien pour la répartition d'automates réactifs que pour l'exploration d'autres champs d'applications qu'il a été décidé de mettre en œuvre les algorithmes de synthèse de réseaux bornés [1] et de réseaux répartissables.

Cela était nécessaire car le caractère abstrait et global des régions rend tout à fait fastidieux l'utilisation manuelle de ces techniques et relativement peu intuitifs les résultats dans ce domaine.

Ce document comporte d'abord un rappel des principes généraux communs à ces questions de synthèse, pour ensuite détailler deux algorithmes :

- l'algorithme de synthèse de réseaux de Petri bornés à partir de systèmes de transition finis.

- Un nouvel algorithme de synthèse n'engendrant que des réseaux répartissables sur un ensemble de processeurs spécifié *a priori* par un placement des actions.

Ces deux algorithmes ont été mis en œuvre dans l'outil SYNET [1], écrit en Caml-light [17].

Une analyse de la complexité de ces algorithmes montre que contrairement aux algorithmes détaillés dans [7, 15, 16] qui ne sont pas polynomiaux (et ne peuvent espérer traiter des problèmes de taille réaliste), les algorithmes mis en œuvre dans SYNET sont polynomiaux en moyenne.

Une application de ces techniques est alors abordée : la synthèse de programmes répartis à partir d'une spécification donnée sous la forme d'un automate. Dans ce contexte les réseaux synthétisés ne sont qu'une représentation de la concurrence et des synchronisations et constituent un intermédiaire à la génération d'automates communicants.

Deux exemples de synthèse de protocoles (exclusion mutuelle et protocole de connexion-déconnexion) permettent l'esquisse d'une méthode fondée sur la synthèse de réseaux distribuables. Cette méthode est radicalement nouvelle en regard des techniques existantes [14, 19, 4].

Les perspectives de recherche indiquent le chemin à suivre pour transformer cette esquisse en véritable méthode, fondée sur des résultats solides et apportant à ces techniques de synthèse de réseaux une première véritable application.

1. Les lecteurs désireux d'utiliser l'outil SYNET peuvent s'adresser à l'auteur : `Benoit.Caillaud@irisa.fr`

2 Automates, régions et réseaux

2.1 Automates

Un *automate* est un système de transition fini, noté $A = (S, E, T, s_0)$ où : S est un ensemble fini d'états ; E un ensemble d'actions, fini et disjoint de S ; $T \subseteq S \times E \times S$ une relation de transition et $s_0 \in S$ l'état initial.

Un automate est dit *déterministe* si et seulement si :

$$\forall s, s', s'' \in S,\ \forall e \in E,\ (s, e, s') \in T \wedge (s, e, s'') \in T \ \Rightarrow\ s' = s''$$

La synthèse de réseaux ne s'applique qu'à des automates déterministes, réduits (chaque action apparaît dans au moins une transition) et accessibles (tous les états sont accessibles depuis l'état initial s_0). Dans ce qui suit nous considérons un automate déterministe, accessible et réduit A ayant n actions et m états.

2.2 Régions

Une *région rationnelle* est un morphisme de A vers $\mathbb{Q}$, c'est à dire une paire d'applications (σ, η) compatible avec la relation de transition :

$$\begin{cases} \sigma : S \to \mathbb{Q} \\ \eta : E \to \mathbb{Q} \end{cases}$$

telle que $\forall (s, e, s') \in T,\ \sigma(s') = \sigma(s) + \eta(e)$.

L'application η est appelée *région abstraite* ou *région flottante*. Une région est dite *entière* si elle prend ses valeurs dans $\mathbb{Z}$ et *positive* si σ est partout positive.

2.3 Chemins et cycles

Notons $\overline{E}$ le *conjugué* de l'ensemble des actions ($\overline{e}$ est le conjugué de l'action e et $\overline{\overline{e}} = e$). Notons également $\overline{T} = \{(s', \overline{e}, s) | (s, e, s') \in T\}$ le conjugué de la relation de transition T. Notons $A^\star$ le *complété* du système de transition A : $A^\star = (S, E \uplus \overline{E}, T \uplus \overline{T}, s_0)$.

Un *chemin* de A est une suite alternée états-actions de la forme :

$$s_0, e_0, s_1, \ldots\ e_{p-1}, s_p$$

telle que $\forall i \in \{0 \ldots p-1\},\ (s_i, e_i, s_{i+1}) \in T$.

Un *cycle* est un chemin finissant sur son état de départ. Deux chemins, dont le second part de l'état final du premier peuvent être *concaténés*.

Une *base de cycles* de $A^\star$ est un ensemble minimal (pour l'inclusion) de cycles engendrant tout les cycles de $A^\star$ par composition (cancaténation et changement d'origine). On obtient une base de cycles de $A^\star$ en considérant un arbre maximal (couvrant) de A et en prenant les plus petits cycles contenant chacun exactement une co-arête [11].

2.4 Vecteurs de Parikh, cycles et régions

Le vecteur de parikh d'un mot de $\overline{E}^\star$ ou d'un chemin de $A^\star$ est un vecteur de $\mathbb{Z}^E$ défini comme suit (1_e dénote le vecteur unité dans la dimension e) :

$$\left\{ \begin{array}{ll} & \pi(\epsilon) = 0 \\ \forall e \in E, \forall u \in \overline{E}^\star & \pi(e \cdot u) = \pi(u) + 1_e \\ & \pi(\overline{e} \cdot u) = \pi(u) - 1_e \end{array} \right.$$

Si nous nous donnons un arbre couvrant de A (déterministe et accessible), il est possible d'associer un vecteur de Parikh aux états de l'automate : c'est le vecteur de Parikh du chemin dans l'arbre allant de s_0 à l'état considéré.

Nous considérons dans ce qui suit un arbre couvrant de A donné. Les vecteurs de Parikh des états sont notés $\pi : S \to \mathbb{Z}^E$ et la base des cycles $\gamma_1, \ldots \gamma_p$.

2.5 Base de cycles, base des régions flottantes

On peut remarquer que l'application $\eta : \mathbb{Q}^E$ est une région flottante si et seulement si pour tout cycle γ de $A^\star$, le produit scalaire $\eta \cdot \pi(\gamma)$ est nul.

Soit C la matrice formée des vecteurs de Parikh de la base de cycles $\gamma_1, \ldots \gamma_p$ considérée :

$$C = \left[\begin{array}{c} \pi(\gamma_1) \\ \vdots \\ \pi(\gamma_p) \end{array} \right]$$

Une région flottante η, définie comme vecteur colonne vérifie :

$$C \cdot \eta = 0$$

C'est à dire que l'ensemble des régions flottantes est le noyau de C. Soit B une base de $\ker C$ — elle est appelée matrice des régions flottantes de base. Les régions flottantes sont donc de la forme : $\eta = B\lambda$ où λ est un vecteur rationnel de dimension $\dim(\ker C)$.

Dans ce qui suit nous supposons que la base B des régions flottantes de A est de dimension q.

3 Algorithmes

Cette partie décrit les algorithmes mis en œuvre dans l'outil SYNET, à savoir :

- l'algorithme de synthèse de réseaux de Petri bornés dont le graphe de marquage est isomorphe à un automate donné [1].
- Une extension de cet algorithme incorporant des contraintes liées à la prise en compte d'un placement des actions sur un ensemble de processus, en vue de la génération de programmes répartis.

3.1 Problème de synthèse, problèmes de séparation

Le problème de synthèse de réseau est de décider constructivement de l'existence d'un réseau de Petri pur, dont l'ensemble des transitions est en bijection avec l'alphabet des actions, et dont le graphe de marquage est isomorphe à un automate donné.

Il a été montré (dans [9] pour les réseaux élémentaires et dans [1] pour les réseaux de Petri bornés) que ce problème admet une solution si et seulement si l'automate est séparé, c'est à dire qu'il satisfait les deux problèmes de séparation suivants :

Séparation état-état Pour toute paire d'états distincts s_1 et s_2 il existe une région flottante η de l'automate qui les sépare : $\eta(s_1) \neq \eta(s_2)$.

Séparation état-action Pour tout état s et toute action e interdite dans cet état, il existe une région positive (σ, η) de l'automate qui les sépare : $\sigma(s) + \eta(e) < 0$.

La procédure de décision consiste à résoudre chacun des problèmes de séparation. Pour chaque problème une région est calculée et le réseau synthétisé est formé de l'ensemble de ces régions. Il est toutefois intéressant de vérifier avant de générer une nouvelle région qu'aucune des régions déjà calculées ne résout le problème de séparation considéré.

3.2 Synthèse de réseaux bornés

Considérons un automate A déterministe, accessible et réduit ; un arbre maximal de A issu de son état initial s_0 ; la base des cycles généralisés de A (définie par l'arbre maximal de A) ; une base B des régions flottantes de A, calculée à partir de la base des cycles généralisés.

Résolution des problèmes de séparation état-état Une région flottante η sépare deux états distincts s_1 et s_2 si et seulement si $\pi(s_1) \cdot \eta \neq \pi(s_2) \cdot \eta$. Autrement dit on cherche un vecteur λ de dimension q tel que :

$$(\pi(s_1) - \pi(s_2))\, B\lambda \neq 0$$

En fait il suffit de considérer la séparation par au moins une région de base. Dans l'outil SYNET, l'ensemble des régions engendrées est initialisé à l'ensemble des régions de base. Les régions de base inutiles sont éliminées à la fin.

Résolution des problèmes de séparation état-action Une région positive (σ, η) sépare l'état s et l'action e si et seulement si : $\sigma(s) + \eta(e) < 0$. Autrement dit :

$$\begin{cases} & \sigma(s_0) + (\pi(s) + 1_e) \cdot \eta < 0 \\ \forall s' \in S & \sigma(s_0) + \pi(s') \cdot \eta \geq 0 \end{cases}$$

On peut transformer ce système d'inéquations par soustraction des autres lignes à la première ligne :

$$\forall s' \in S,\ (\pi(s) - \pi(s') + 1_e) \cdot \eta < 0$$

Ce système est homogène. Il admet une solution si et seulement si le système suivant en admet une :

$$\forall s' \in S,\ (\pi(s) - \pi(s') + 1_e) \cdot \eta \leq -1$$

C'est à dire on cherche λ vecteur rationnel tel que :

$$\begin{bmatrix} \pi(s) - \pi(s_1) + 1_e \\ \vdots \\ \pi(s) - \pi(s_m) + 1_e \end{bmatrix} B\lambda \leq \begin{bmatrix} -1 \\ \vdots \\ -1 \end{bmatrix}$$

avec $\sigma(s_0) = -\min_{s' \in S} \pi(s') B\lambda$ et $\eta = B\lambda$.

Dans l'outil SYNET, la résolution des problèmes de séparation état-action se fait par la résolution dans $\mathbb{Q}$ du programme linéaire suivant :

$$\min \left\{ \Sigma_i \lambda_i | \begin{bmatrix} \pi(s) - \pi(s_1) + 1_e \\ \vdots \\ \pi(s) - \pi(s_m) + 1_e \end{bmatrix} B\lambda \leq \begin{bmatrix} -1 \\ \vdots \\ -1 \end{bmatrix} \right\}$$

L'algorithme utilisé est la méthode du Simplexe dans les rationnels [21]. La région produite est ensuite ramenée dans $\mathbb{Z}$ par multiplication par le plus petit dénominateur commun.

3.3 Synthèse de réseaux distribuables

Placement des actions. Nous nous donnons, en plus d'un automate, un placement des actions sur un réseau de processus $I : \zeta : E \to I$. Ce qui nous donne un placement des transitions du réseau engendré.

Réseaux distribuables. Le problème de la répartition des réseaux de Petri a été abordé pour la première fois dans [12]. Un réseau est dit réparti quand chacune de ses transitions et places est localisée sur un processeur d'une architecture répartie. La principale difficulté est la mise en œuvre des conflits répartis, c'est à dire des places ayant un flot sortant vers au moins deux transitions qui ne sont pas localisées sur le même processus. Des schémas de répartition de réseaux de Petri ont été proposés : ils ne peuvent malheureusement répartir que des classes très particulières de réseaux sans pour autant garantir une quelconque optimalité de la concurrence extraite et des communications.

Nous adoptons ici une démarche radicalement différente : nous faisons en sorte de n'engendrer que des réseaux distribuables — sans conflits répartis. Chaque place ne peut être décrémentée que par des transitions localisées sur un seul et même processeur. Les places qui ont cette propriété sont dites *localisables*. La *localisation* d'une place localisable se déduit de son flot sortant : c'est le processeur propriétaire des transitions décrémentant la place [2].

La communication entre processus se résume alors à assurer une certaine causalité entre événements et ne porte pas sur la résolution de conflits.

2. Une place ayant un flot sortant nul peut être localisée sur n'importe quel processeur.

Résolution des problèmes de séparation état-état Le problème est que les régions flottantes de base ne sont pas utilisables pour résoudre les problèmes de séparation état-état — elles ne sont pas nécessairement localisables.

Pour calculer une région localisable séparant les états s_1 et s_2, il faut résoudre l'inéquation suivante, avec des contraintes de localisation supplémentaires :

$$(\pi(s_1) - \pi(s_2))\, B\lambda \neq 0$$

La contrainte de localisation est qu'il existe un processus $i \in I$ tel que la région $\eta = B\lambda$ est positive pour toute action qui n'est pas localisée sur i. Ceci peut être exprimé par une contrainte linéaire de la forme :

$$L_i B\lambda \leq 0$$

Sans perte de généralité, ceci revient à trouver une solution pour l'un des deux systèmes d'inéquations suivants :

$$\begin{bmatrix} L_i \\ \pi(s_1) - \pi(s_2) \end{bmatrix} B\lambda \leq \begin{bmatrix} 0 \\ -1 \end{bmatrix}$$

ou :

$$\begin{bmatrix} L_i \\ \pi(s_2) - \pi(s_1) \end{bmatrix} B\lambda \leq \begin{bmatrix} 0 \\ -1 \end{bmatrix}$$

Comme pour la séparation état-action, l'algorithme utilisé dans SYNET est la méthode du Simplexe.

Résolution des problèmes de séparation état-action Une région, localisable sur $i \in I$, positive (σ, η) sépare l'état s et l'action e si et seulement si $\sigma(s) + \eta(e) < 0$, avec une contrainte de localité de la forme :

$$L_i \eta \leq 0$$

Autrement dit, exactement comme pour la séparation état-action dans le cas général, il faut résoudre le système suivant :

$$\begin{bmatrix} L_i \\ \pi(s) - \pi(s_1) + 1_e \\ \vdots \\ \pi(s) - \pi(s_m) + 1_e \end{bmatrix} B\lambda \leq \begin{bmatrix} 0 \\ -1 \\ \vdots \\ -1 \end{bmatrix}$$

avec $\sigma(s_0) = -\min_{s' \in S} \pi(s') B\lambda$ et $\eta = B\lambda$.

À nouveau, l'algorithme utilisé est la méthode du Simplexe [21].

3.4 Élimination des régions redondantes

Les problèmes de séparation sont résolus les uns après les autres. Pour chaque problème le réseau synthétisé peut s'enrichir d'une nouvelle place (région). Cette place peut rendre redondantes une ou plusieurs places déjà engendrées. De ce fait le réseau synthétisé n'est pas minimal pour l'inclusion. Il s'agit maintenant de supprimer les places redondantes.

Une place est redondante si en la retirant du réseau on ne change pas le comportement de ce dernier. Autrement dit elle ne résout aucun problème de séparation qui n'est résolu par l'ensemble des autres places. Elle peut alors être retirée du réseau et l'on poursuit la minimisation avec le réseau moins la place redondante.

Cette élimination des redondances est très importante : elle garantit la minimalité (pour l'inclusion) du réseau synthétisé.

3.5 Complexité

Complexité en moyenne de l'algorithme du simplexe Le nombre moyen de pivotages effectués sur un système de c contraintes sur v variables est $\mathcal{O}(\min(c, v, c-v))$ — voir chapitre 11 de [21]. Chaque pivotage a une complexité $\mathcal{O}(cv)$.

Notations Notons n le nombre d'actions de l'automate, m son nombre d'états et l son nombre de transitions. Notons q le nombre de régions flottantes de base et s le nombre de problèmes de séparation état-action. Pour la synthèse de réseaux répartissable, nous considérons k processus.

L'arbre couvrant de l'automate possède $m-1$ arêtes. Chaque cycle de base est défini par une co-arête, il y en a donc $l+1-m$.

Notons r le nombre de régions synthétisées par la résolution des problèmes de séparation.

Arbre couvrant, base des cycles Le calcul de l'arbre couvrant se fait par parcours en profondeur du graphe. Chaque transition est visitée et à chaque fois un nouveau vecteur de Parikh est calculé — soit le vecteur de Parikh d'un état visité pour la première fois, soit pour le calcul du vecteur de Parikh d'un cycle de base. La complexité est donc :

$$c_1 = \mathcal{O}(ln)$$

Base de régions abstraites Le calcul se fait par élimination gaussienne. Le nombre de pivotages est au plus $2\min(l+1-m, n)$. La complexité est donc :

$$c_2 = \mathcal{O}(\min(l+m, n)(l+m)n)$$

Séparation état-état La séparation état-état se fait par produit scalaire des vecteurs de Parikh avec les vecteurs de base (complexité $\mathcal{O}(qmn)$), puis par comparaison de ces produits scalaires entre eux : $\mathcal{O}(qm^2)$.

$$c_3 = \mathcal{O}(qm(m+n))$$

Dans le cas de la synthèse de réseaux distribuables, il faut résoudre des programmes linéaires. La taille de ces programmes linéaires est au plus n contraintes et q variables. Leur nombre est $\mathcal{O}(km^2)$, ce qui donne la complexité suivante[3] :

$$c_3^d = \mathcal{O}(km^2nq\min(n,q))$$

Séparation état-action Dans le cas sans contrainte de répartition, chaque problème linéaire possède m contraintes et q variables :

$$c_4 = \mathcal{O}(smq\min(m,q))$$

Avec contrainte de répartition, le nombre de programmes linéaires est au plus multiplié par $\mathcal{O}(k)$ et la taille des programmes linéaires est au plus $m+n$ contraintes de q variables :

$$c_4^d = \mathcal{O}(sk(m+n)q\min(m+n,q))$$

Minimisation L'élimination des régions redondantes consiste à calculer les problèmes de séparation résolus par chacune des régions calculées et de décider si une région résout au moins un problème de séparation qui n'est résolu par aucune des autres régions.

Le nombre de régions synthétisées r est majoré par $q+mn-l$. Le nombre de problèmes de séparation est $m(m-1)/2+mn-l$. La décision de résolution d'un problème de séparation par une région se fait en temps constant. De ce fait la décision de redondance d'une région a pour complexité :

$$\mathcal{O}(r(m(m+n)-l))$$

La complexité de la minimisation est donc :

$$c^m = \mathcal{O}(r^2(m(m+n)-l))$$

En fixant $q = m = n = \mathcal{O}(l)$ et en utilisant la majoration de r donnée ci-dessus nous obtenons une complexité dans le pire des cas : $\mathcal{O}(l^6)$. Si on fixe $q = n = \mathcal{O}(1)$ et $m = \mathcal{O}(l)$ la complexité dans le pire des cas est : $\mathcal{O}(l^4)$.

Complexités en moyenne Nous considérons la classe des automates à l transitions. En moyenne nous avons : $n = m = p = q = s = \mathcal{O}(l)$. La complexité moyenne de la synthèse de réseaux de Petri bornés sans minimisation est : $c = \mathcal{O}(l^4)$.

Pour la synthèse de réseaux distribuables (sans minimisation et en supposant $k = \mathcal{O}(l)$), la complexité moyenne est $c^d = \mathcal{O}(l^5)$.

Cependant, si on considère le nombre de processus et le nombre d'actions constant ($k = n = \mathcal{O}(1)$), sans minimisation du réseau synthétisé on obtient les complexités :

$$c_{n=\mathcal{O}(1)} = c^d_{n=k=\mathcal{O}(1)} = \mathcal{O}(l^2)$$

3. Nous supposons que les opérations arithmétique sur les rationnels se font en temps $\mathcal{O}(1)$.

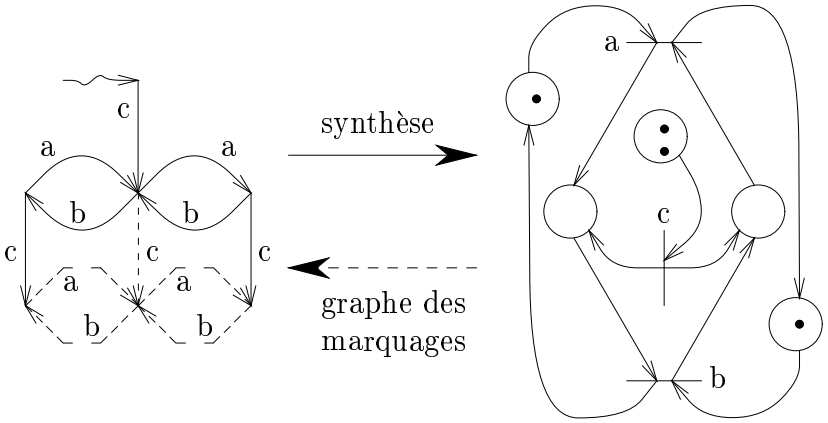

FIG. 1: *Automate ne satisfaisant pas la propriété de séparation état-action*

Interprétation En calculant la complexité avec un nombre d'événements et de processus constant ($n = k = \mathcal{O}(1)$) ou croissant comme la taille du graphe ($n = k = \mathcal{O}(l)$), nous montrons que ces algorithmes de synthèse sont très sensibles au nombre d'événements de l'automate.

On peut cependant remarquer que les spécifications de service des protocoles de communication comportent généralement peu d'événements (en regard de la taille de l'automate) et que pour celles-ci l'approximation $n = k = \mathcal{O}(1)$ est probablement meilleure que $n = k = \mathcal{O}(l)$.

3.6 Exemples

L'exemple de la figure 1 montre un automate qui ne satisfait pas trois problèmes de séparation état-action. Le graphe des marquages du réseau engendré est une extension [4] de l'automate.

Le deuxième exemple (figure 2) montre comment la synthèse de réseau peut être contrainte par un placement des actions. Dans ce cas , les régions négatives sur a et b sont interdites alors que la synthèse sans contrainte peut produire un réseau comportant une place ayant un flot sortant vers les transitions a et b.

4 Synthèse d'automates communicants

4.1 Introduction

Le problème est ici d'utiliser les informations obtenues par synthèse de réseau distribuable pour synthétiser un système d'automates communicants (par compteurs [18] ou par files) qui soit observationnellement équivalent à l'automate de départ. Ces informations portent sur les synchronisations (régions) qu'il faut réaliser sous la forme de communications asynchrones. Le problème et les techniques utilisées (à l'exception bien sûr de la synthèse de réseaux) sont proches de celles utilisées dans [5].

4. Dans le cas où les états sont séparés, l'automate est inclus dans le graphe des marquages du réseau synthétisé.

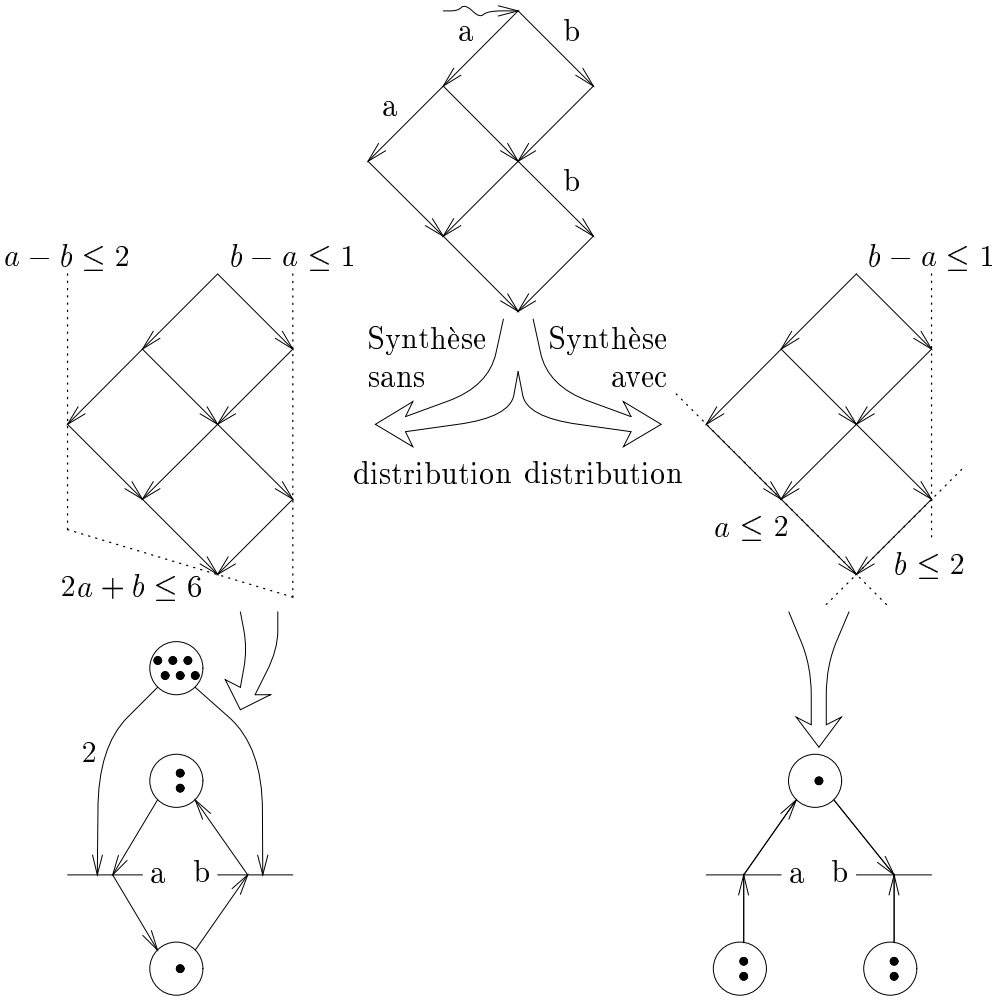

FIG. 2: *Synthèse avec et sans contrainte de répartition*

4.2 Principe

Soit $A = (S, E, T, s_0)$ un automate déterministe, séparé pour un placement des actions $\zeta : E \to I$. Soit X un ensemble de places (ou régions) localisables selon ζ et séparant A.

Soit $\xi : X \to I$ une localisation des places. Elle vérifie :

$$\forall (\sigma, \eta) \in X, \forall e \in E, \zeta(e) \neq \xi(\sigma, \eta) \Rightarrow \eta(e) \geq 0$$

Notons Y l'ensemble des places de communication :

$$Y = \{(\sigma, \eta) | (\sigma, \eta) \in X, \exists e \in E, \zeta(e) \neq \xi(\sigma, \eta) \text{ et } \eta(e) \neq 0\}$$

Le système d'automates communicants par compteurs est formé d'un automate par site de I et d'un compteur par place de communication (ensemble Y). Chaque compteur est localisé là où est localisée sa place associée. Les automates peuvent incrémenter n'importe quel compteur ; il ne peuvent cependant décrémenter que leurs compteurs locaux. Ce compteurs peuvent être mis en œuvre par des canaux de communication sans perte, ce qui permet de retrouver le modèle

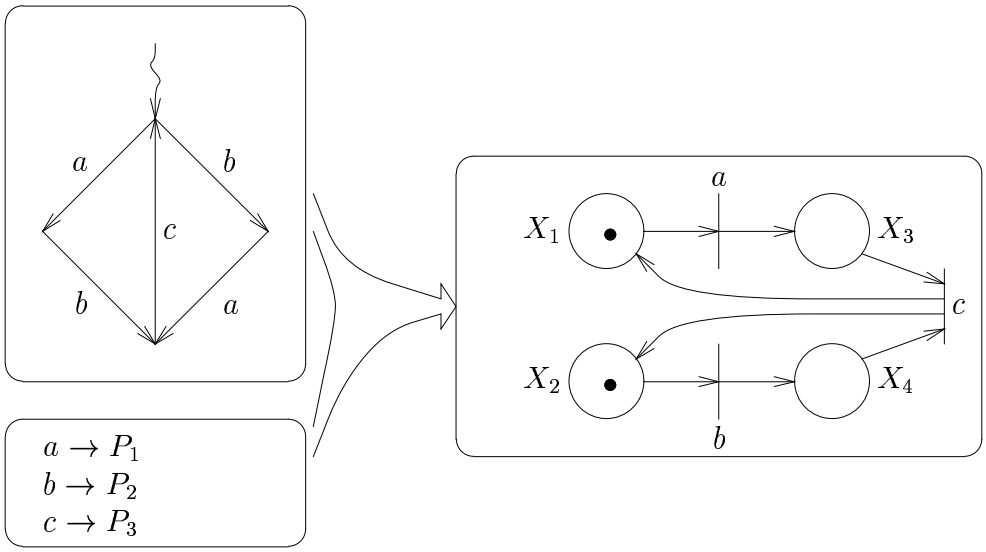

FIG. 3: *Automate, placement des actions et réseau associé*

bien connu des automates communicants. Les messages échangés sont alors d'un seul type et représentent les jetons du réseau.

L'automate communicant du site $i \in I$ est obtenu par récriture des étiquettes des transitions de l'automate A :

– Une action e, locale ($\zeta(e) = i$) est récrite en :

$$\{?|\eta(e)|(\sigma,\eta)|(\sigma,\eta) \in Y \text{ and } \eta(e) < 0\} \cup \{e\} \quad \cup$$
$$\{!|\eta(e)|(\sigma,\eta)|(\sigma,\eta) \in Y \text{ and } \eta(e) > 0\}$$

– Une action e, non locale ($\zeta(e) \neq i$) est récrite en action silencieuse τ.

On obtient alors une collection d'automates dont les transitions sont étiquetées soit par un ensemble de réceptions, une action de E et un ensemble d'émissions de messages ; soit par une action silencieuse τ. Il convient alors de minimiser ces automates modulo bisimulation faible.

On peut remarquer que les envois de messages se font aussitôt que possible et les réceptions le plus tard possible. D'autres stratégies de communication sont possibles : réceptions au plus tôt, émissions au plus tôt ou réception au plus tard et émission au plus tard.

4.3 Exemple

Prenons l'exemple de la figure 3 : les actions a et c sont placées sur le processus P_1 ; l'action b est placée sur le processus P_2. Dès lors, les régions ρ_1, ρ_3 et ρ_4 sont localisées sur P_1 et la région ρ_2 est localisée sur P_2. Toutes les régions sont locales à l'exception de ρ_2 et ρ_4. Ainsi nous obtenons le système d'automates communicants de la figure 4, qui après minimisation donne le système de la figure 5.

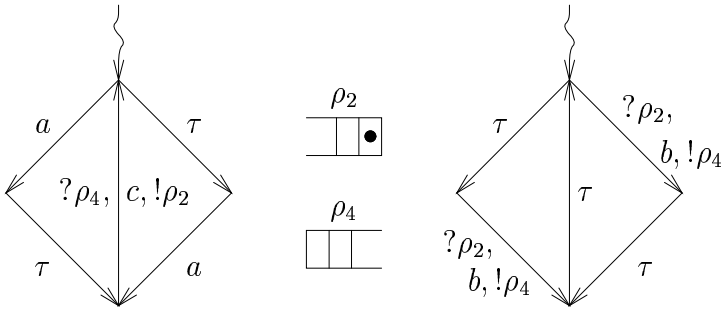

FIG. 4: *Système d'automates communicants non minimisé*

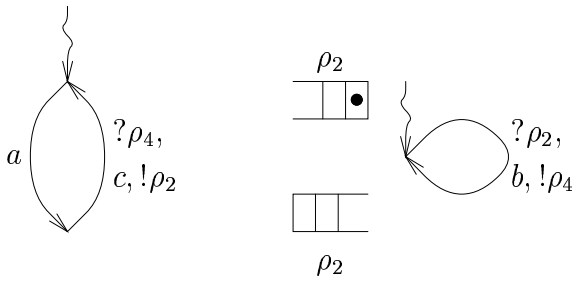

FIG. 5: *Minimisation du système d'automates communicants*

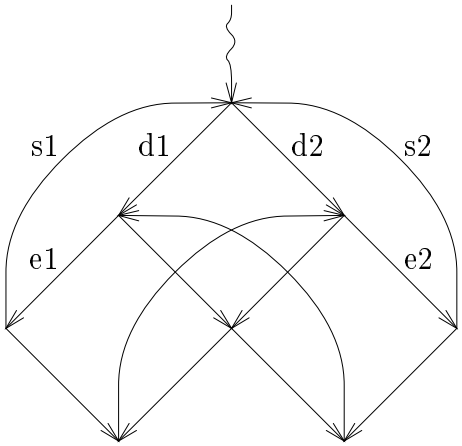

FIG. 6: *Service de l'exclusion mutuelle*

5 Exemples de répartitions

5.1 Exclusion mutuelle

Considérons deux entités (1 et 2) pouvant chacune effectuer une demande d'entrée en section critique ($d1$ pour l'entité 1 et $d2$ pour l'entité 2), une entrée en section critique ($e1$ et $e2$) et une sortie de section critique ($s1$ et $s2$). Le service d'exclusion mutuelle entre les entités 1 et 2 est défini par l'automate de la figure 6.

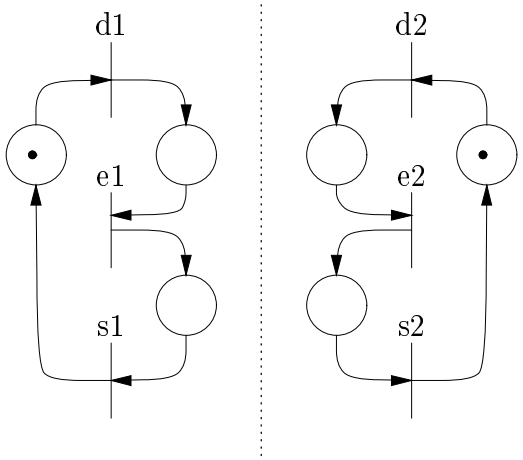

FIG. 7: *Réseau engendré, non séparant*

Nous effectuons la synthèse de réseau distribuable avec deux processus : les actions $d1$, $e1$, $s1$ sont placées sur le processus P_1 et $d2$, $e2$, $s2$ sur le processus P_2.

> L'automate de la figure 6 n'est pas séparé pour cette distribution : Les comportements $d1\,d2\,e1\,e2$ et $d2\,d1\,e2\,e1$ ne peuvent être interdits — i.e. la propriété d'exclusion mutuelle est violée. Le réseau synthétisé comporte deux processus séquentiels autonomes (sans interaction) de la figure 7.

Le problème provient du conflit réparti entre $e1$ et $e2$. Il y a deux solutions pour localiser le conflit :

1. La création d'un troisième processus effectuant le choix entre $e1$ et $e2$: c'est un *contrôleur*. Cela conduit au service modifié de la figure 8. Les actions $\tau 1$ et $\tau 2$ sont localisées sur le contrôleur.

2. La circulation d'un *jeton* entre les deux processus, donnant tour à tour le choix à chacun des processus. Le service ainsi modifié est donné figure 11. Les actions $\tau 1$ et $\tau 2$ représentent l'échange de ce jeton.

Première solution Considérons le service raffiné de la figure 8. Les actions sont placées comme suit :

- $d1$, $e1$, $s1$ sur le processus P_1,
- $d2$, $e2$, $s2$ sur le processus P_2,
- $\tau 1$ et $\tau 2$ sur le contrôleur.

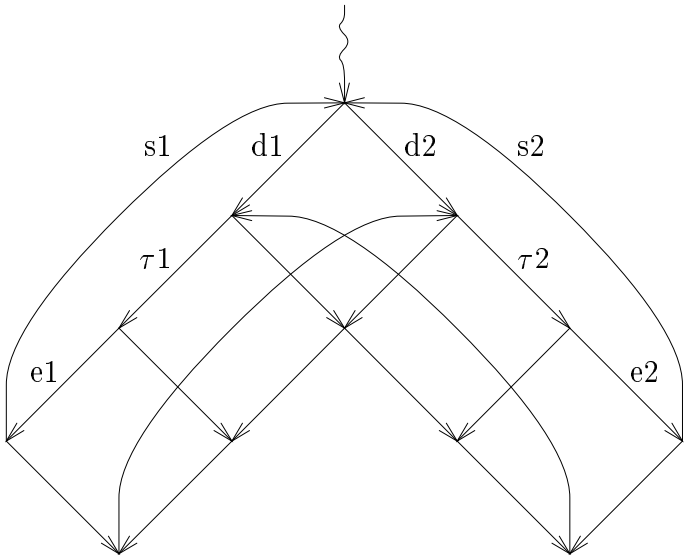

FIG. 8: *Premier raffinement*

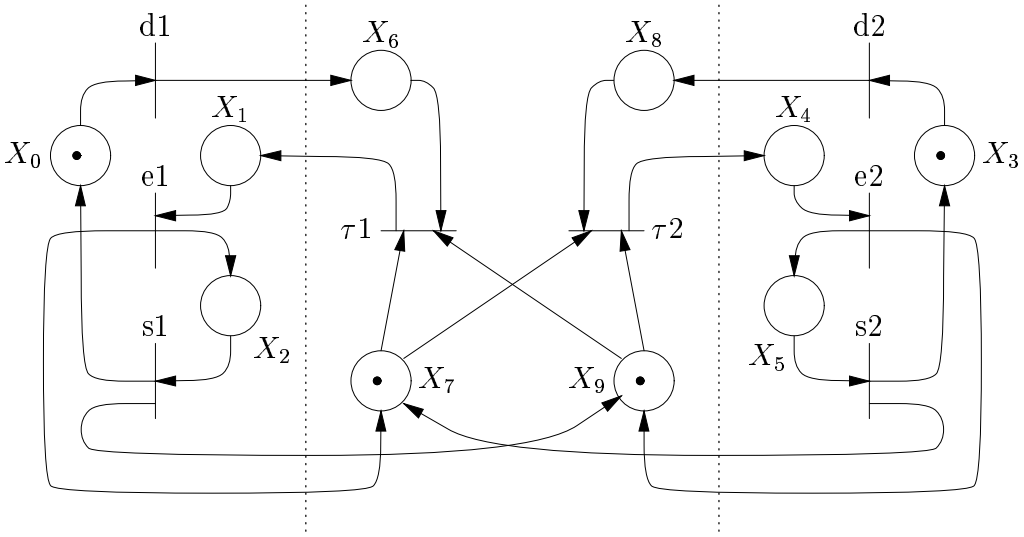

FIG. 9: *Réseau distribuable engendré*

Cet automate est séparé pour la synthèse de réseaux distribuables. Le réseau produit est donné figure 9 : à gauche se trouve le processus P_1, à droite le processus P_2 et le contrôleur est au centre.

Pour engendrer un système d'automates communicants il suffit de considérer comme communications les incréments et décréments de places de communication — émission pour le processus propriétaire de l'action et réception pour le processus propriétaire de la place.

Après minimisation, cela donne les automates de la figure 10.

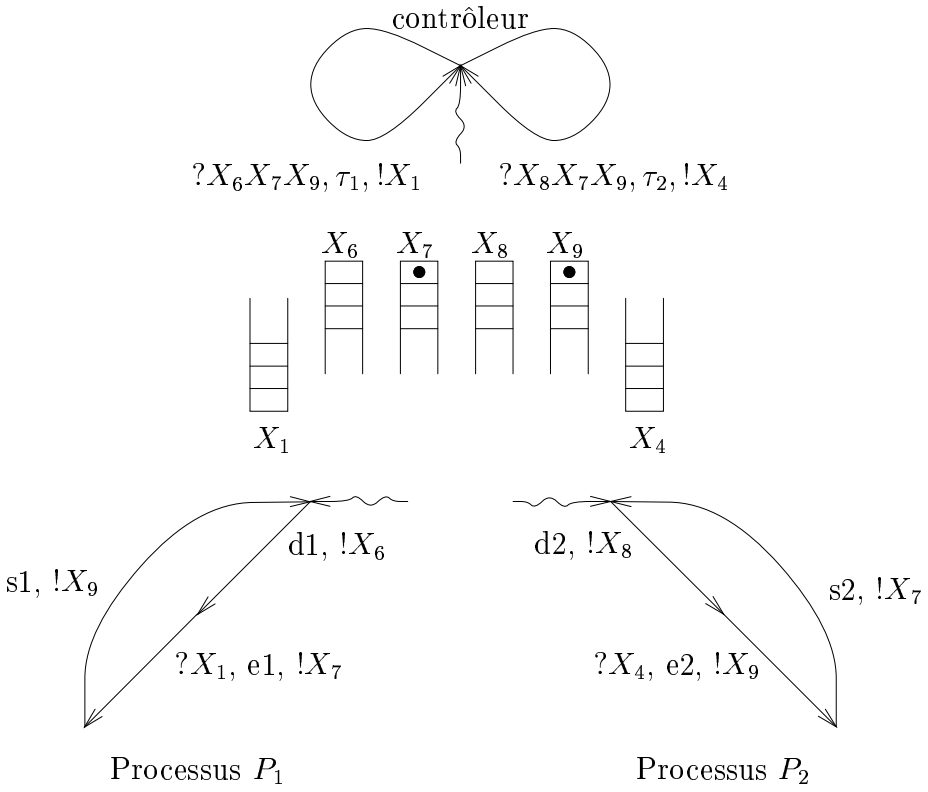

FIG. 10: *Protocole du contrôleur et des processus P_1 et P_2*

Deuxième solution Considérons le service raffiné de la figure 11. Les actions $\tau 1$ et $\tau 2$ sont placées respectivement sur l'entité 1 et 2. Les autres actions sont placées comme précédemment.

Avec ce placement, la synthèse de réseaux distribuables indique que l'automate est séparé et produit le réseau de la figure 12.

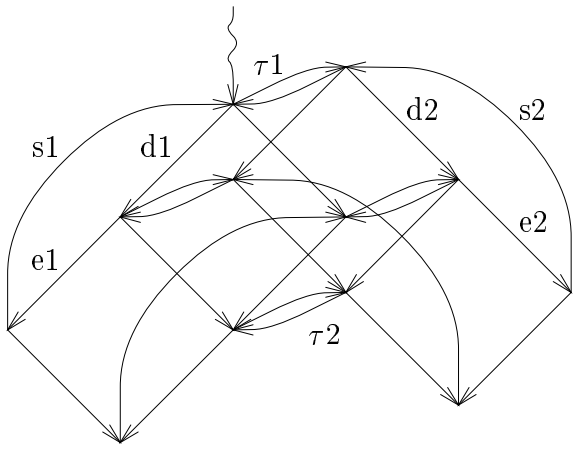

FIG. 11: *Deuxième raffinement*

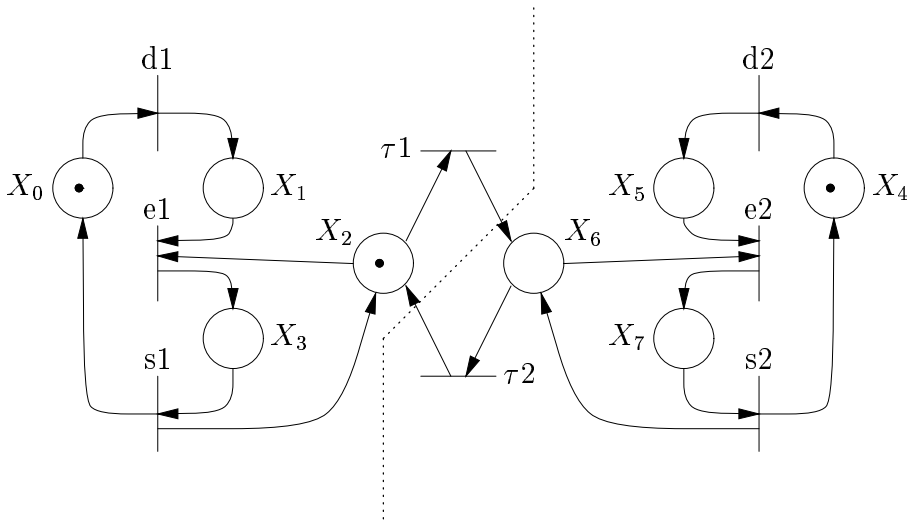

FIG. 12: *Réseau distribuable engendré*

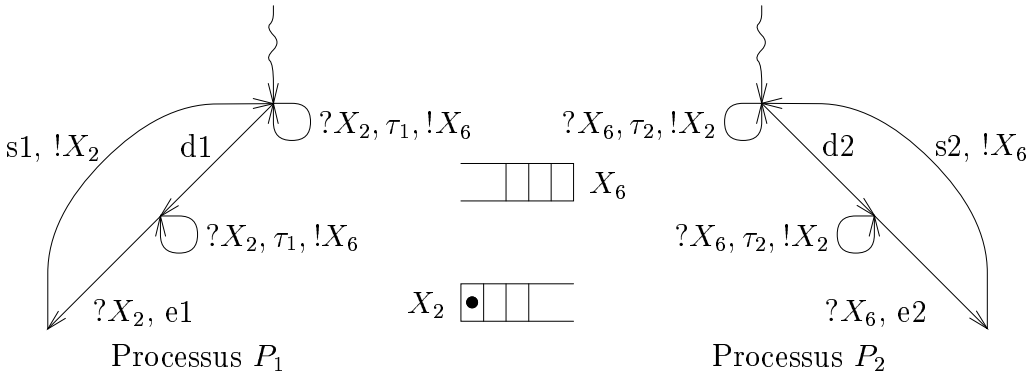

FIG. 13: *Protocole des processus P_1 et P_2*

Le système d'automates communicants est donné figure 13.

5.2 Connexion-déconnexion

Spécification de service Le protocole de connexion-déconnexion [13] a pour fonction de gérer une suite de sessions entre deux entités (A et B) communiquant au travers d'un réseau supposé fiable. L'entité A peut demander l'établissement (a) ou la fermeture (b) d'une connexion avec B. L'entité B ne peut demander que la fermeture de la connexion (c). Le protocole doit assurer que entre deux a il y a toujours un b ou un c, au plus un b et au plus un c ; entre deux b il y a un a et entre deux c, il y a un a. Cette propriété est exprimée par l'automate de la figure 14.

Le problème de la synthèse d'un protocole de connexion-déconnexion est de produire un programme réactif réparti en deux processus communicants A et B observationnellement équi-

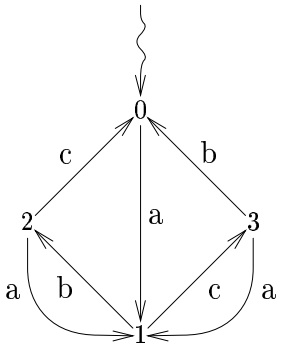

FIG. 14: *Service de connexion-déconnexion*

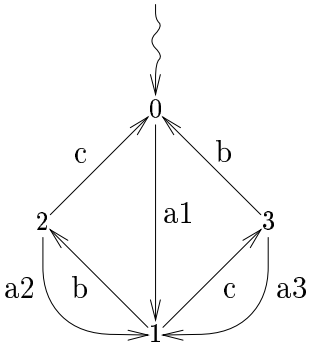

FIG. 15: *Premier raffinement*

valent à la spécification de service et dont les actions visibles sont réparties comme suit : a et b sont placées sur A, c est placée sur B.

> L'algorithme de synthèse de réseau distribuable échoue sur la spécification du service (figure 14) : les états 0, 1, 2 et 3 sont indiscernables. Le réseau produit ne comporte aucune place.

Il faut en effet remarquer que l'automate n'est pas co-déterministe, puisque 3 transitions étiquetées a arrivent en 1. C'est la cause principale de l'échec de la synthèse.

Premier raffinement Le premier raffinement consiste à rendre l'automate co-déterministe.

> Pour cela, toute occurrence de a est remplacée par $a1$, $a2$ ou $a3$ — figure 15. ces trois actions sont placées sur l'entité A.

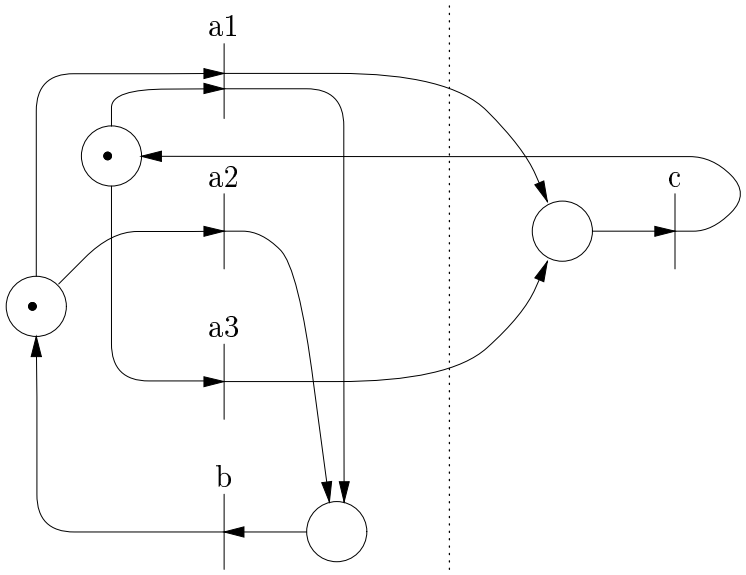

FIG. 16: *Premier raffinement, réseau engendré, non séparant*

La synthèse de réseau distribuable (figure 16) échoue également sur cet automate.

> L'automate de la figure 15 n'est pas séparé : les actions $a2$ et $a3$ sont tirables en 0.

La cause de cet échec est ici plus subtile. On peut remarquer que à cause de la transition $(3, b, 0)$, toute région (σ, η) de l'automate satisfait $\sigma(3) = \sigma(0) - \eta(b) \geq 0$. Le cycle $a_2 b$ fait que $\eta(b) = -\eta(a_2)$. On a alors : $\sigma(0) + \eta(a_2) \geq 0$. Il n'est donc pas possible de séparer l'état 0 et l'action a_2 — idem pour a_1.

Deuxième raffinement Le deuxième raffinement consiste à s'affranchir de la contrainte liant $a1$, $a2$ et $a3$. Cela peut être réalisé en rallongeant les cycles $(1, b, 2, a2, 1)$ et $(1, c, 3, a3, 1)$.

> Le raffinement produit l'automate de la figure 17.
>
> – Des transitions silencieuses étiquetées $\tau 1$ et $\tau 2$ sont insérées devant les transitions étiquetées $a2$ et $a3$.
>
> – les états desquels partent les transitions étiquetées $a2$ et $a3$ sont équivalents, on peut donc les fusionner.
>
> – Pour localiser les conflits $(\tau 1, b)$, et $(\tau 2, c)$, les actions $\tau 1$ et $\tau 2$ sont respectivement placées sur A et B.

L'automate de la figure 17 est séparé ; l'algorithme de synthèse produit le réseau distribuable de la figure 18.

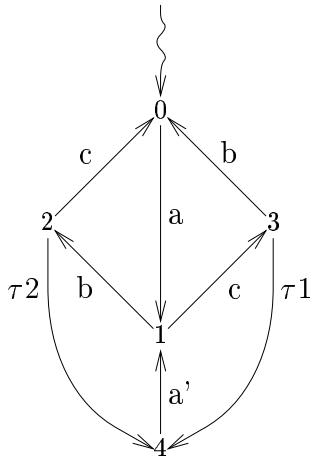

FIG. 17: *Deuxième raffinement*

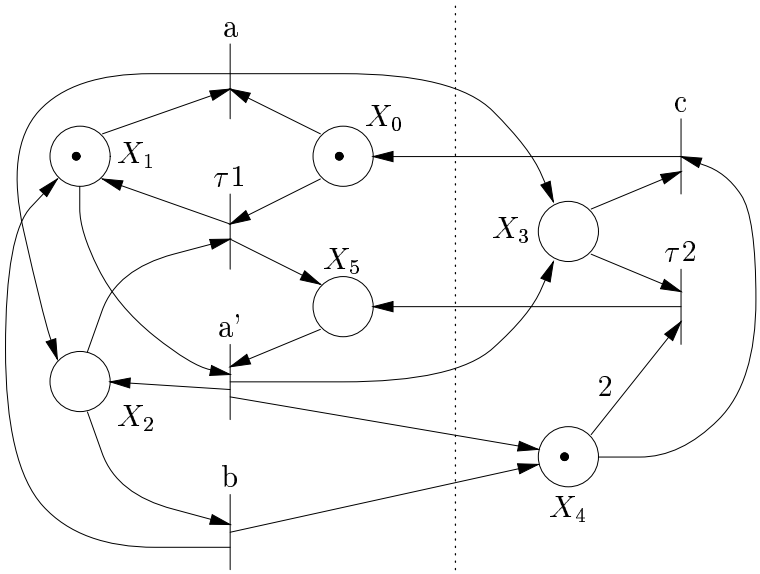

FIG. 18: *Réseau distribuable*

Expression du protocole par des automates communicants Après avoir synthétisé un réseau distribuable qui réalise le service voulu, il faut maintenant produire un programme réparti comportant deux automates communicants, pour les entités A et B.

Appliquons la méthode exposée précédemment :

- Les régions X_0 et X_5 sont des entrées de l'extrémité A.
- Les régions X_3 et X_4 sont des entrées de l'extrémité B.

Les automates communicants des extrémités A et B sont donnés figure 19.

Quelques comportements typiques du protocole sont donnés dans la figure 20. On peut remarquer que ces comportements sont exactement ceux des implantations du service de connexion-

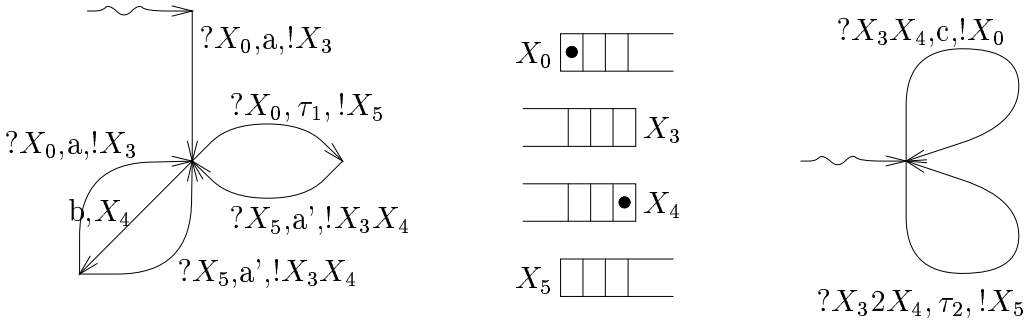

FIG. 19: *Protocole de connexion-déconnexion, extrémités A et B*

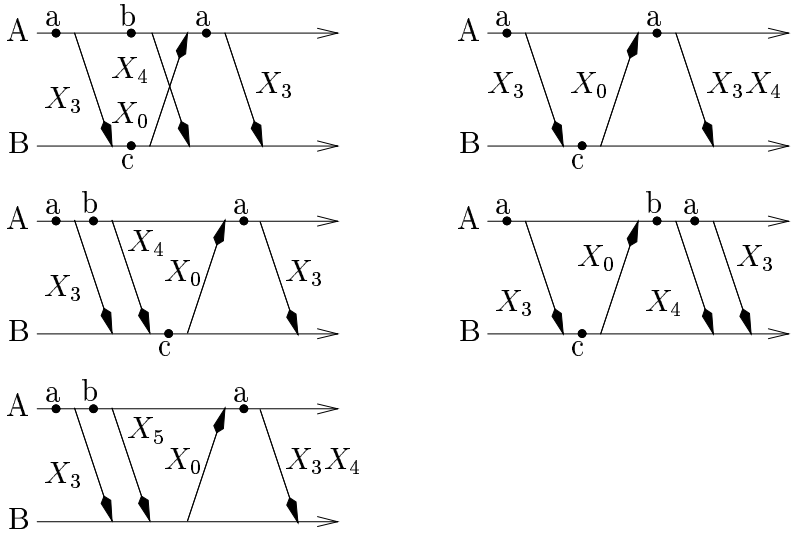

FIG. 20: *Protocole de connexion-déconnexion : comportements nominaux*

déconnexion décrits dans la littérature [14] à l'exception du message X_4 dans le comportement $a\,c\,b\,a$.

6 Perspectives de recherche

Le principal frein pour l'application de la synthèse de réseaux aux problèmes de synthèse de protocoles de communication est la non-séparation de leurs spécifications de services. L'utilisateur de l'outil SYNET doit alors transformer ces spécifications en scindant certaines actions, en dépliant l'automate ou en ajoutant des transitions silencieuses. Même si la correction de ces raffinements est facile à établir, ils n'ont pas pour autant de justification formelle et la démarche requiert l'expertise de l'utilisateur. Serait-il possible de retranscrire les échecs de séparation en heuristiques de raffinement qui guideraient l'utilisateur dans son travail de manipulation de spécifications? La question est ouverte.

Les algorithmes de synthèse de réseaux peuvent-ils être étendus? Dans quelles directions faut-il étendre l'outil SYNET? Suggérons ici quelques pistes de recherche ou d'applications :

- La synthèse de réseaux dont le langage est encadré par deux langages réguliers donnés. Ceci permettrait la spécification imprécise d'un service par encadrement entre propriété de vivacité et propriété de sûreté.
- la complexité des algorithmes de synthèse est telle (dans les cas les plus défavorables, puissance 5 de la taille du système de transition) que leur utilisation sur des spécifications de plus de mille transitions n'est pas envisageable. Une parallélisation de ces algorithmes permettrait de gagner au plus un ordre de grandeur sur la taille des graphes traités. Seule l'introduction d'une certaine modularité dans l'approche laisse espérer l'application de ces techniques à des spécifications de grande taille. On peut effectivement décomposer la synthèse de réseaux à partir d'un produit partiellement synchronisé de systèmes de transition finis. Cela a-t'il un intérêt pratique?
- Un dernier axe de recherche est l'augmentation de l'expressivité des réseaux synthétisés par l'outil SYNET :
  - synthèse de réseaux non bornés,
  - ou de réseaux auto-modifiants [3].
- Les évolutions à venir de l'outil SYNET porteront sur son intégration dans des environnements de validation (Open/Caesar [10]) et de manipulation de réseaux (Design/CPN [8]).

Références

[1] E. Badouel, L. Bernardinello, and P. Darondeau. Polynomial algorithms for the synthesis of bounded nets. In P. D. Mosses, M. Nielsen, and M. I. Schwartzbach, editors, *TAPSOFT'95: Theory and Practice of Software Development*, volume 915 of *Lecture Notes in Computer Science*, pages 364–378, Aarhus, Denmark, May 1995.

[2] E. Badouel and P. Darondeau. Dualities between nets and automata induced by schizophrenic objects. Rapport de recherche 2452, Institut de Recherche en Informatique et en Automatique, mars 1995.

[3] E. Badouel and P. Darondeau. Stratified petri nets. Publication interne 1092, IRISA, mars 1997.

[4] D. Brand and P. Zafiropulo. Synthesis of protocols for unlimited number of process. In *Proc. of Trends Applications, Comp. Network Protocols Symp. , Gaithersburg*, pages 29–40, May 1980.

[5] B. Caillaud. Contribution à la modélisation du SPMD : distribution asynchrone d'automates. Thèse de doctorat de l'université de Rennes 1, No 1126, juin 1994.

[6] J. Cortadella, M. Kishinevsky, and A. Kondratyev. Complete state encoding based on the theory of regions. In *2nd International Workshop on Advanced Research in Asynchronous Circuits and Systems*, pages 36–47, 1996.

[7] J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Synthesizing petri nets from state-based models. In *Proc. of ICCAD'95*, pages 164–171, 1995.

[8] CPN-Group. *Design/CPN Online*. University of Aarhus, Denmark. "http://www.daimi.aau.dk/designCPN/".

[9] A. Ehrenfeucht and G. Rozenberg. Partial (set) 2-structures, parts 1 and 2. *Acta Informatica*, 27:315–368, 1990.

[10] H. Garavel. *The Open/Caesar Reference Manual*. VERIMAG, March 1995.

[11] M. Gondran and M. Minoux. *Graphes et algorithmes*. Eyrolles, Paris, 1985.

[12] R. Hopkins. Distributable nets. In G. Rozenberg, editor, *Advances in Petri Nets 1991*, volume 524 of *Lecture Notes in Computer Science*. Springer-Verlag, 1991.

[13] C. Jard and M. Raynal. Specification of properties is required to verify distributed algorithms. In *Colloque AFCET sur les langages de spécification*, février 1987.

[14] C. Kant Antonescu. Synthèse de spécifications de protocoles à partir de spécifications de service. Thèse de doctorat, université de Montréal, mars 1993.

[15] M. Kishinevsky, J. Cortadella, A. Kondratyev, L. Lavagno, A. Taubin, and A. Yakovlev. Place chart nets and their synthesis. Technical Report 96-2-003, Department of Computer Hardware, The University of Aizu, November 1996.

[16] M. Kishinevsky, J. Cortadella, A. Kondratyev, L. Lavagno, and A. Yakovlev. Synthesis of general petri nets. Technical Report 96-2-004, Department of Computer Hardware, The University of Aizu, November 1996.

[17] X. Leroy. *The Caml Light system, documentation and user's guide*. Institut national de recherche en informatique et en automatique, version 0.73 edition, January 1997. "http://pauillac.inria.fr/caml/man-caml/index.html".

[18] G. Lesventes. Systèmes d'automates à compteurs et semi-linéarité des ensembles d'états accessibles : "forlorn hope". Thèse de doctorat de l'université de Rennes 1, No 336, 1989.

[19] K. Saleh and R. Probert. Synthesis of error-recoverable protocol specifications from service specifications. In *International Conference on Computing and Information Niagara Falls, Canada, LNCS # 468*, May 1990.

[20] V. Schmitt. Flip-flop nets. Publication interne 936, Institut de recherche en informatique et systèmes aléatoires, June 1995.

[21] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, 1986.

Unit´e de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unit´e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unit´e de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unit´e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unit´e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399